→ code
→ Imp definitions
→ Imp. Heading

Date ___/___/_____

Data Structures & Algorithms

Syllabus of Data Structures :→ Basic DS → Array, Strings etc.

Operations on Linear DS:
(*) Access, Insertion, Deletion, Traversal, Search, Sort, Update, Merge, Memory Management

1. Linked List - all
2. Stack, Queue - all
3. Searching & Sorting & Hashing & Collision Resolution & Storage Management
4. Binary Trees - all, + Heaps.
5. Graphs

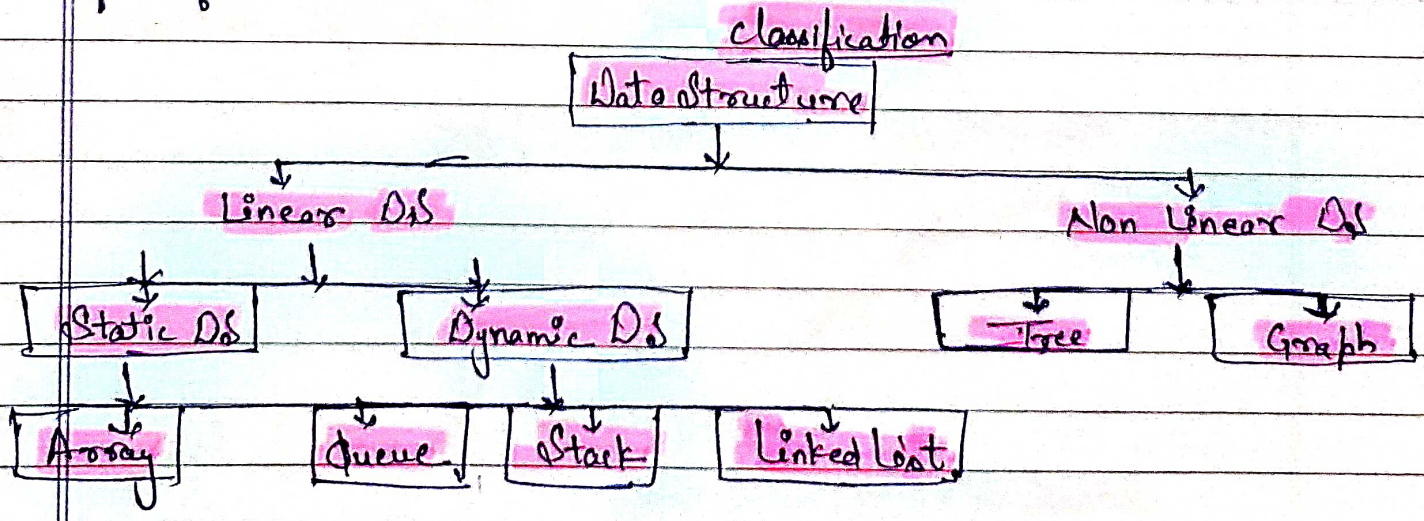
Operations on Non Linear DS.
(*) : Tree Traversal, Graph Traversal, Heap Operations, Balancing.

Methods & Algorithms.

1. Recursion & Backtracking
2. Greedy
3. Dynamic Programming
4. NP & NP complete & NP hard.

{C++ STL.}
{Java Collection framework}

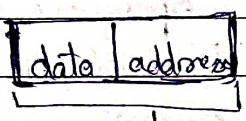
Data-Structure → Data structures involves Organization, Storage, Manipulation of data. They provides systematic way to store and manage information, they are building blocks of algorithms and cumical part of Software Development.



Date ___ / ___ / _____

Linked List

Whenever we don't have enough contiguous space to store a large number of elements together then, we have a solution. The Linked list Data structure is used to store data in non-contiguous format having section to store data and address inside a node and nodes are the unit block of any type of linkedlist they have a data section and address section to store address of previous & next node.



node

Array vs Linked list

Saatvik

Date ___ / ___ / _____

Types of Linked List → (I) Singly LL (II) Doubly LL
(III) Circular LL (IV) Circular Doubly LL

(I) **Singly LL** →

Node →

```
class Node{
public:
int data;
Node* ptrNext;
};
Node :: Node(int data){
this->data= data;
this-> ptrNext = NULL;
}
```

Insertion of Node into Head position.

```
void InsertAtHead( Node* &head, int d){
Node* temp = new Node(d);
temp->ptrNext = head;
head = temp; }
```

Insertion of Node in tail position.

```
void insertATtail( Node* & tail, intd){
Node* temp = new Node(d);
tail-> next = temp;
tail = temp;
}
```

print function for Linked list

```
void print(Node* & head){
Node* temp = head;
while( temp != NULL){ cout<< temp->data
tem = temp-> next; }
```

Date ___ / ___ / _____

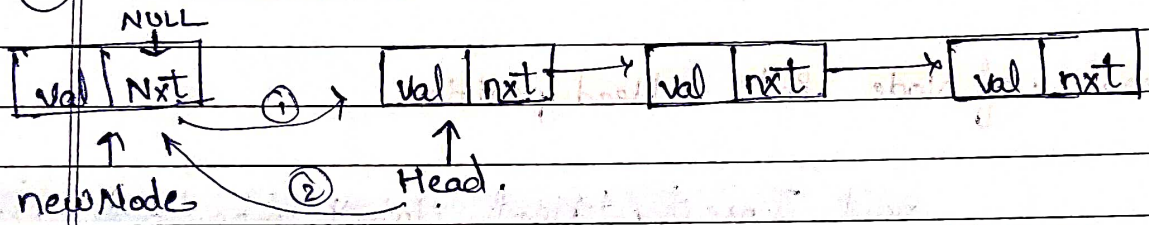
Destructor for class Node:

```
~Node(){
    int val = this -> data;
    if(this -> next != NULL){
        delete next;
        this -> next = NULL;
    }
    Cout << " Deleting the node with the value " << val << "\n";
}
```

Sol →

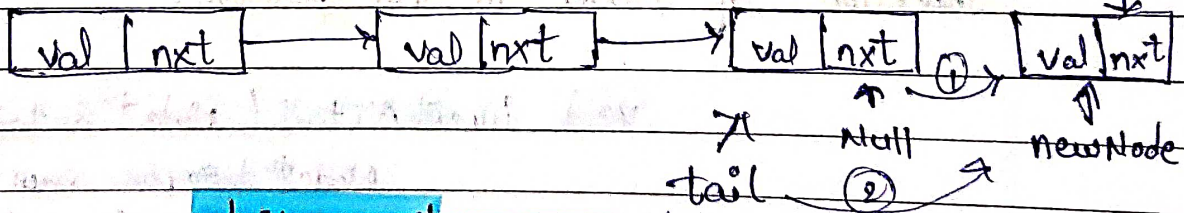
node type!

(i) Insertion at Head.




```
newNode -> next = Head;
Head = newNode;
```

(ii) Insertion at END



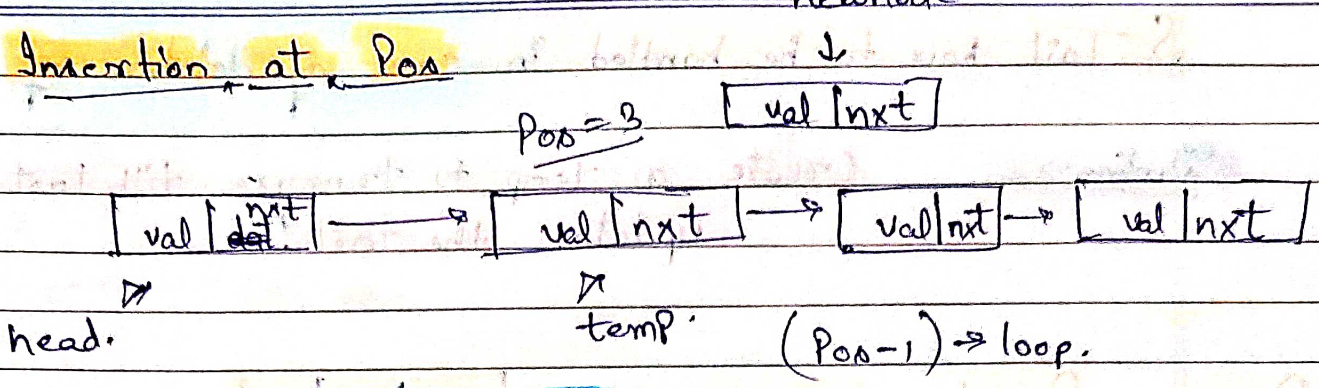

```
tail -> next = newnode
tail = newnode.
```

Page No. ____

Date ___/___/_____

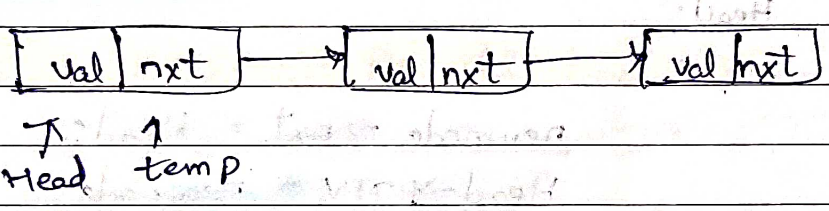
(Saathi)

(3) Insertion at Pos




```
temp->next = newNode;
newnode->next = temp->next
```

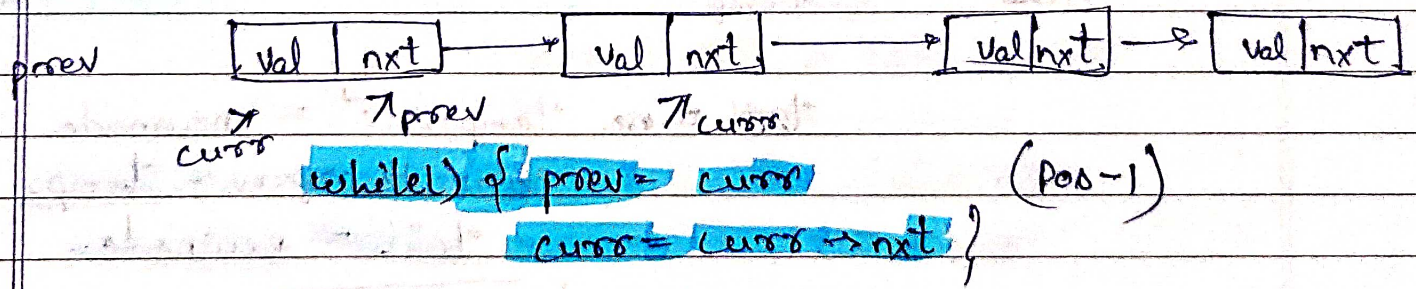
* Deletion at Head



{ * Destructor }

```
Node *temp = Head;
head = head->next;
temp->next = NULL;
delete temp;
```

Deletion of node at POS:




```
while() { prev = curr
curr = curr->nxt; }

prev->next = curr->next;
curr->next = NULL;
delete curr;
```

Page No. ____

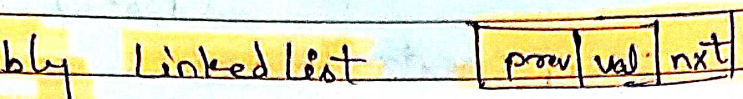
Saathi

Date ___/___/_____

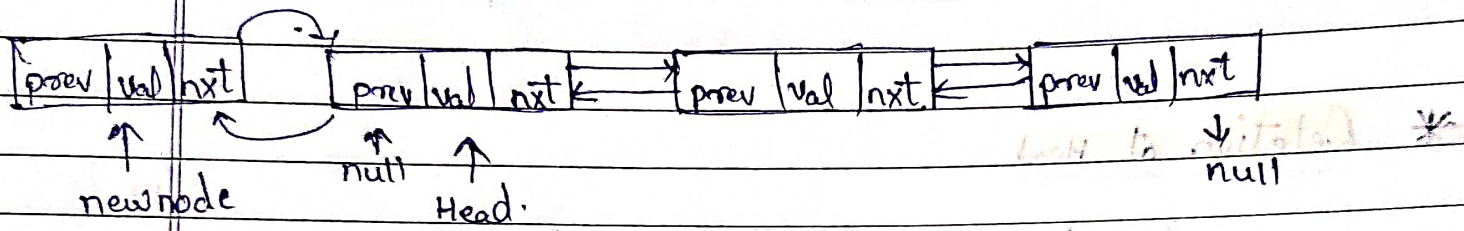
§ Tail have to be handled in case of deletion of last node.

Solution → Create a temp to traverse till last. update the tail.

DLL → Doubly Linked list

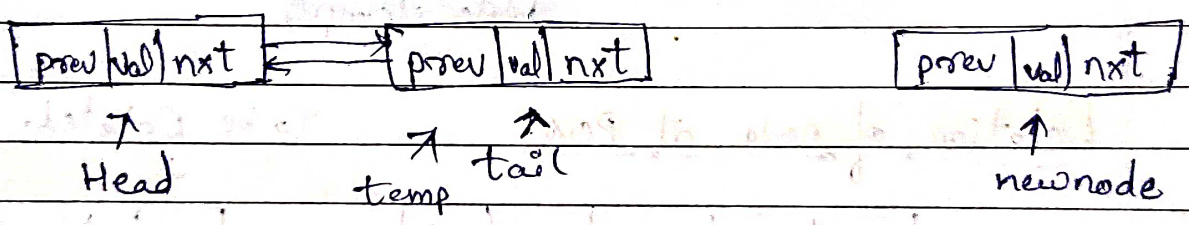


insertion at Head



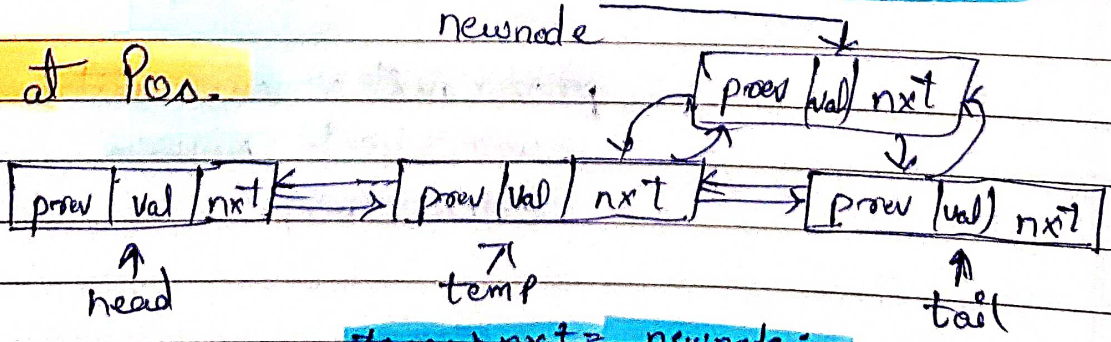

```
newnode->nxt = Head;
Head->prev = newnode;
newnode Head = newnode;
```

insertion at Last




```
tail = temp->nxt = newnode
newnode->prev = temp
tail = newnode.
```

insertion at Pos.



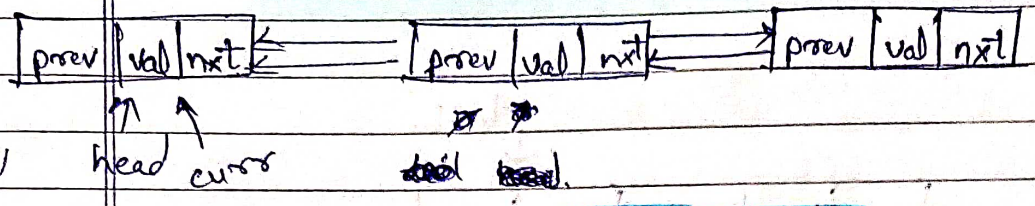

```
temp->nxt = newnode;
newnode->prev = temp;
temp->nxt->prev = newnode;
```

Page No.

Saathi

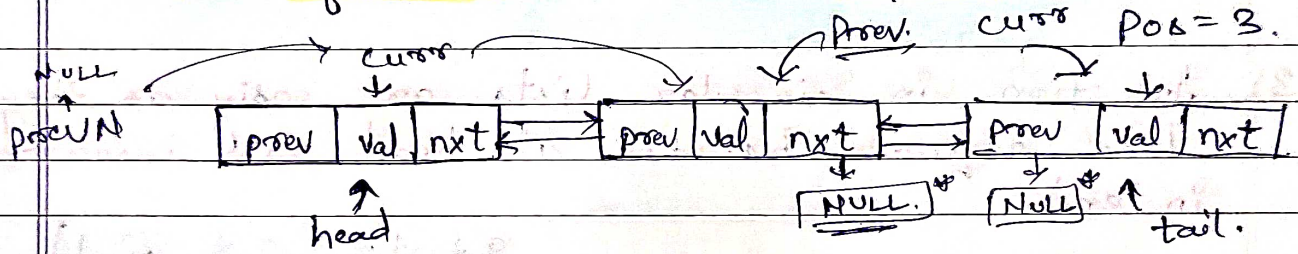
Date ___ / ___ / ____

* Deletion of Node at head...




```
temp = head;
head = head -> nxt;
head -> next -> prev = NULL;
temp -> next = NULL;
delete temp;
```

* Deletion of Node at Pos && Last / tail.



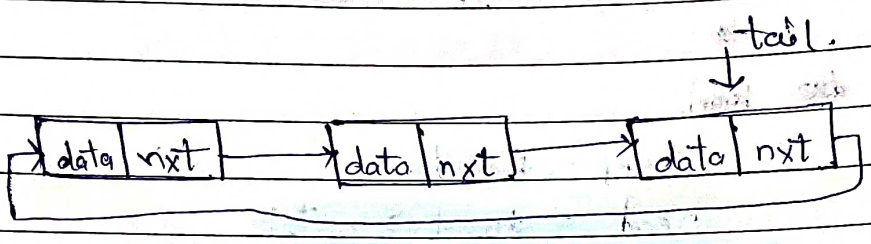

```
while( cnt < pos) {
    prev = curr;
    curr = curr -> nxt;
    cnt++;
}
prev -> next = curr -> nxt
curr -> nxt = NULL
curr -> next -> prev = prev
curr -> next = NULL
curr -> prev = NULL
delete curr;
```

Page No. ____

Saathi

Date ___/___/_____

Singly Circular Linked List →



(1) We donot need any head pointer as it is a circular. So in order to find entry point we use a pointer i.e. (tail here).

(2) the end of the list is not exist in circular. List.

3) Insertion in circular lists are easy as they donot require insertion at head or insertion in end.

```
void insertionOfNode(Node* &tail, int data, int find)
{
    if (tail == NULL){
    Node* Newnode = new Node(data);
        tail = Newnode;
        Newnode->next = tail;
    }
    else{
        Node* temp = new Node(data);
        Node* curr = tail;
        while (curr->data != find){
            curr = curr->next);
        }
    temp->next = curr->next.
    curr->next = temp;
```

Page No.

Date ___/___/_____

Saathi

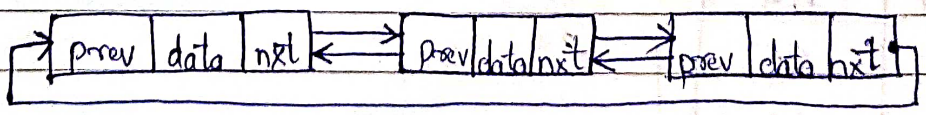
Deletion :->

```
void deleteNode(Node* &tail, int value){
    if(tail == NULL){
        cout<<"List is empty"<<endl;
        return;}
    else{
        Node* prev = tail;
        Node* curr = prev->next;
        while(curr->data != value){
            prev= curr;
            curr = curr->next;}
        prev->next = curr->next;
        if(tail == curr){
            tail= prev;}
        curr->next =NULL;
        delete curr;
    }
```

optimization :->

```
if(curr == prev){
    tail=NULL;}
```

Circular Doubly Linked List.



{ * Same as Previous Circular Linked List * }

=> Approach to solve Linked List question.

(i) While()&& Loop method. (ii) Slow & Fast pointer

(iii) Recursion Method.